Vagrant - Eine eigene Vagrant-Box erstellen

1

Zuerst wird des Betriebssystem in einer neuen VM installiert. In meinem Fall hab' ich das aktuelle Oracle Linux 7.4 von der Delivery Cloud (https://edelivery.oracle.com/) heruntergeladen.

Die neue VM trägt den Namen "ol74-image". Da die künftigen VMs über ein privates Netzwerk angesprochen werden sollenl, ist ein zweiter Netzwerkadapter nötig. Die übrigen Einstellungen können auf den Standardwerten verbleiben, der Arbeitsspeicher kann z. B. pro per Vagrant erstellte VM gewählt werden.

Das NAT-Netzwerk kann nicht entfernt werden! Vagrant erwartet bzw. erstellt auf dem ersten Adapter ein NAT-Netzwerk. Da ich meine erste Box ohne NAT erstellen wollte, hat es mich einen Tag gekostet , bis ich auf diese Tatsache stoß:

> " eth0 as NAT is a fundamental requirement of Vagrant in its current state. Perhaps in the future this limitation can be removed, but for the time being it is required." - https://github.com/hashicorp/vagrant/issues/2093

Das Betriebsystem wird mit "minimaler Installation" installiert. Nach der Installation existiert nur der root-Benutzer (kein Netzwerk oder zusätzliche Software).

2

Nun soll die neue Maschine (und damit auch künftige) kommunizieren können. Nach dem Reboot erfolgt also die Konfiguration des Netzwerks. Der erste Adapter wird als NAT mit DHCP konfiguriert, der zweite für das private Netzwerk.

```
nmcli dev status
nmcli con add type 802-3-ethernet ifname enp0s3 con-name
enp0s3 autoconnect yes ipv4.method auto
nmcli con add type 802-3-ethernet ifname enp0s8 con-name
enp0s8 autoconnect yes ipv4.method manual ipv4.addresses
„192.168.56.10/24“
nmcli dev status

reboot

nmcli dev status
nmcli dev show | less
```

Nach einem Test-Reboot werden die Adapter als "verbunden" angezeigt.

```
GERÄT   TYP       STATUS          VERBINDUNG
enp0s3  ethernet  verbunden       enp0s3
enp0s8  ethernet  verbunden       enp0s8
```

Die Ausgabe von "show" zeigt mehr Infos, wie z. B. die IP-Adressen. Auch ein Ping vom Host-Comupter ist erfolgreich:

```
PING 192.168.56.10 (192.168.56.10): 56 data bytes
64 bytes from 192.168.56.10: icmp_seq=0 ttl=64 time=0.
320 ms
64 bytes from 192.168.56.10: icmp_seq=1 ttl=64 time=0.
311 ms
^C
--- 192.168.56.10 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet
loss
round-trip min/avg/max/stddev = 0.311/0.316/0.320/0.004
ms
```

3

Um Datenaustausch mit dem Host und der virtuellen Maschine möglich zu machen, sind die VB-Gasterweiterungen notwendig. Über das Menü von Virtual Box "Devices - Insert Guest Additions CD image..." wird die CD mit den Gasterweiterungen eingelegt. Installiert werden sie folgendermaßen:

```
mount -r /dev/cdrom /mnt
/mnt/VBoxLinuxAdditions.run
```

Auf einem jungfräulichen System wie dem verwendeten klappt dies leider nicht so einfach. Es fehlen noch ein paar Pakete. So schlägt der erste Aufruf des Installers fehl mit:

```
bzip2: Kommando nicht gefunden
```

Ist dieses Paket installiert, bekommt man eine Meldung in der Art: "Es sieht so aus, als wären die Gasterweiterungen schon installiert." Fährt man trotzdem fort, schlägt die Installation wieder fehl:

```
Failed to set up service vboxadd
```

Im zweiten Logfile sieht man auch, warum das so ist:

```
unable to find the sources of your current Linux kernel
```

Nun ist also die Paketliste komplett und nach deren Installation können auch die VB-Erweiterungen erfolgreich geladen werden. Dazu muss vorher allerdings die Umgebungsvariable KERN_DIR auf das Verzeichnis mit den geladenen Sourcen gesetzt werden. Am einfachsten geht dies über die TAB-Verfolständigung.

```
yum install kernel-uek-devel bzip2 wget
export KERN_DIR=/usr/src/kernels/4.1.12-112.14.1.el7uek.
x86_64
/mnt/VBoxLinuxAdditions.run
..
vboxadd.sh: Starting the VirtualBox Guest Additions.
```

Das Paket "wget" macht später den Import des Vagrant-Keys leichter. Daher installiert man das am besten hier gleich mit. Außerdem sollte wget auf keinem Linux-System fehlen .

4

Jetzt wird der Vagrant-User angelegt. Mit diesem User verbindet sich vagrant standardmäßig und führt Befehle aus. Der User muss Befehle per sudo ausführen können, dies wird in der sudoers-Datei eingetragen.

```
useradd --no-user-group -g wheel vagrant
echo "vagrant" | passwd --stdin vagrant


visudo
Zeile ergänzen: vagrant ALL=(ALL) NOPASSWD:ALL
```

Damit alle Kommandos auf der virtuellen Maschine ausgeführt werden können, muss eine SSH-Verbindung per Key möglich sein. Hierzu wird der "insecure vagrant key" verwendet.

```
mkdir -p /home/vagrant/.ssh
chmod 0700 /home/vagrant/.ssh

wget --no-check-certificate https://raw.github.com
/mitchellh/vagrant/master/keys/vagrant.pub -O /home
/vagrant/.ssh/authorized_keys

chmod 0600 /home/vagrant/.ssh/authorized_keys
chown -R vagrant /home/vagrant/.ssh
```

5

Das System ist nun soweit fertig. Es wird aufgeräumt. Damit das Komprimieren beim Bpx-Packing besser funktioniert, sollte die virtuelle Disk "ausgenullt" werden. Bei meinem Test war die Box-Datei nach dem Ausnullen um 400 MB kleiner.

```
sudo dd if=/dev/zero of=/EMPTY bs=1M
sudo rm -f /EMPTY
yum clean all
history -c
reboot
```

Nach dem Reboot der Maschine kann auf dem Hostcomputer die Box gepackt werden:

```
vagrant package --base ol75-image --output ol75-image.box
```

Wenn man ein Vagrantfile mit hineinpacken will, ergänzt man:

```
--vagrantfile Vagrantfile
```

6

Um die Box nun verwenden zu können, muss sie nur noch in vagrant hinzugefügt werden:

```
vagrant box add ol75-image.box --name ol75-image
```